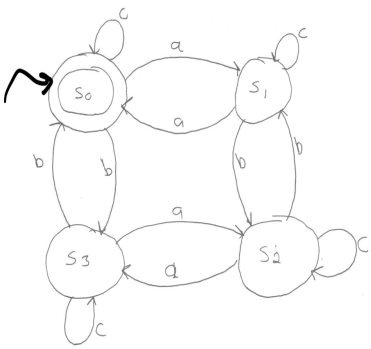
CMPT 125: Assignment #9

Sarbjot Mann
2904949140983741761048971y7980817591052ikjhFUCKYOU39418510515

Simon Fraser University — March 28, 2019

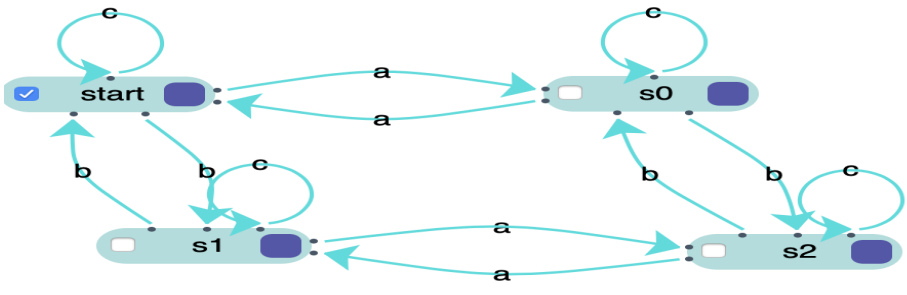
Question 1

Let $\sum = a, b, c$ be the alphabet for this part. Draw an FSM bubble diagram which accepts the language of all strings with an even total number of a's and an even total number of b's. Both must be even. The number of c's can be anything. And before you ask, yes, 0 is an even number. Examples: Your machine should accept: abcabc, acca, abbcbba, ccccc, empty string λ, but reject: abc, cccaccc, bbabb.



We consider s_0 to be our accepting state. If there is no a's or no b's it'll accept the string or accept the string of only c's. However if one a is found it'll go to s_1 and stay there until we get an even a or get an odd b, (since c loops there). If we get an odd b we'll travel to s_2 and it'll stay there unless if we get an even number of b or a. Now we'll either have an even number of b's or a and if we do get them we'll travel back to the accepting state other wise we'll reject the string. The same applies when we start off with a b but in the other order.

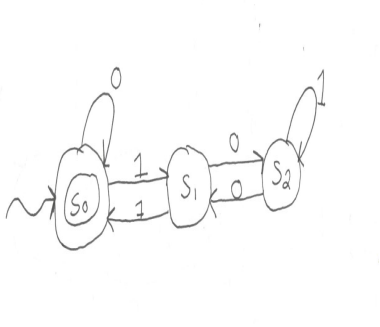
Below is the same FSM but it shows us accepting the correct states (With all cases passing) and rejecting cases that don't satisfy the FSM (with those cases passing as well)



Test Results:
Accept: abcabc -- Pass
Accept: acca -- Pass
Accept: abbcbba -- Pass
Accept: ccccc -- Pass
Accept: [Empty String] -- Pass
Reject: abc -- Pass
Reject: cccaccc -- Pass
Reject: blab -- Pass

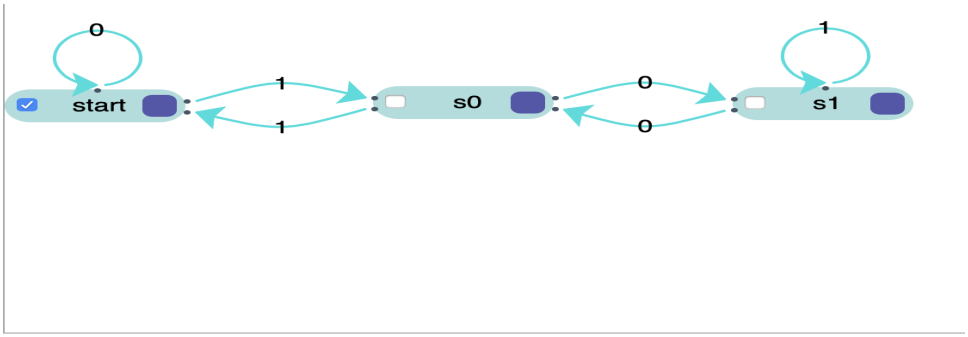
Question 2

Let $\sum = 0, 1$be the alphabet for this part. Draw an FSM bubble diagram which accepts the language of all binary strings which represent integers evenly divisible by 3. Thus, your machine should accept 0, 11, 110, 1001, 1100, 1111, 10010, etc. We won't be fussy about leading 0's on your integers, so you have the option to accept or reject 00, 011, and also empty string λ.



We consider s_0 to be our accepting state.
First let's consider for s_0 to be when the remainder is zero, s_1 for when the remainder is one, and s_2 when the remainder is two. A length of n zeroes is in binary is 0 in decimal form, so lets loop zero on our accepting state. 01, 0000...01, is 1 in decimale form which is not divisible by three so lets more it to s1. However if we have n length of zeros followed by two ones. that is three so lets go back to the accepting state if we see a zero there. N numbered of zeroes followed by 10 is 2, so if we see a zero, lets move it to s2. After, adding one to the end of 10 gives us remainder of two, so lets loop there and if we see a zero go back to s1.

Below is the same FSM but it shows us accepting the correct states (With all cases passing) and rejecting cases that don't satisfy the FSM (with those cases passing as well)




```
Test Results:
Accept: 0 -- Pass
Accept: 11 -- Pass
Accept: 110 -- Pass
Accept: 1001 -- Pass
Accept: 1100 -- Pass
Accept: 1111 -- Pass
Accept: 10010 -- Pass
Accept: 00 -- Pass
Accept: 011 -- Pass
Accept: [Empty String] -- Pass
Reject: 1 -- Pass
```

Question 3

Write a POSIX-style regular expression that represents all lines that begin with b or B and are followed by 0 or more characters.

We know that x|y means either x or y, - means a single character and * means repeat 0... n times

$$\hat{} \quad (b|B).^* \$$$

Alternatively the following is also correct

$$\hat{} \quad [b|B].^* \$$$

(The space between ând the next item is to make it clearer since it looked as if ŵas on top of the brackets)

Question 4

Write a POSIX-style regular expression that represents all lines that contain a word that ends in ion, but not the word "ion" by itself.

The line must contain a word such that it has one word ending with ion but not ion itself. However it doesn't say what all the other elements in the words are. So i'm going to be present two cases.

Case 1: All words
Let's us say only words are in this sentence (potentially). According to posix syntax [:alpha:] represents all uppercase and lowercase letters. Therefore:

$$\hat{} \quad \backslash<[:alpha:]^{*}\backslash> \backslash<[:alpha:]^{+}ion \backslash>^{+}\backslash<[:alpha:]^{*}\backslash> \quad \$$$

Case 2: Characters
Let's us say we have characters in this sentence (potentially). According to posix syntax . represents all characters. Therefore:

$$\hat{} \quad .^{*} \backslash<[:alpha:]^{+}ion \backslash>^{+} .^{*}\backslash> \quad \$$$

Again, the space is just there to make it clearer.